TURNING THE R CONSOLE INTO AN INTERACTIVE LEARNING ENVIRONMENT WITH SWIRL

by
Nicholas A. Carchedi

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science

Baltimore, Maryland
September, 2014

Contents

1 Abstract

Interactive platforms for learning computer programming have exploded in recent years. Sadly, the R programming language is overlooked by most of these tools. swirl is an open source software package for the R programming language that allows users to learn R and statistics interactively, right in the R console. swirl supports multiple questions types, but emphasis is on having the user interact directly with the R prompt, while receiving helpful and immediate feedback. We (swirl's creators) have pioneered early content creation, but swirl is constructed in such a way that anyone can create his or her own content and share it freely with others. While swirl has been downloaded over 70,000 times in its first nine months, development of the software and instructional content is ongoing. We hope that the R and statistics communities will continue to rally around the platform and truly embrace it as their own.

Primary Reader: Jeff Leek, PhD

Secondary Reader: Brian Caffo, PhD

2 Acknowledgements

swirl would not be what it is today without help from some very special people.

First, I would like to thank Bill Bauer and Gina Grdina for their unwavering support. Fortunately, I met Bill and Gina early on in swirl's development. They have made significant contributions to the swirl source code and developed our very popular Regression Models course. I consult with them regularly when I find myself faced with a coding or software design problem that is over my head. Most importantly, they have become good friends and trusted advisers and I consider myself very lucky to have found them.

I'd also like to thank Sean Kross, now a good friend who has made important contributions to the software. In particular, he is the reason that students are able to install swirl courses from a variety of sources with such ease. He has recently begun developing instructional content for swirl and I look forward to continuing our always adventurous collaboration.

In the summer of 2013, soon after I started writing the first version of swirl, I was introduced to Lauren Williams and Ethan Schwartz. Lauren and Ethan were interning in our department while on break from school and helped me develop the very first swirl courses. They were a sounding board for my crazy ideas and swirl would probably not exist in its current form had they not encouraged me to continue developing it.

One name, Hadley Wickham, will appear often in this paper. Hadley is hands down the most prolific R programmer of my generation and he has had an immeasurable impact on the R and data communities. The swirl software depends heavily on several of his R packages and I regularly consult his *Advanced R Programming* book (Wickham 2014a) for answers and advice. Hadley and I have been in touch since swirl first came on the scene and he has always been supportive of the endeavor. He planted in my mind the importance of creating an *authentic* learning environment and that has become central to everything we do with swirl.

I am tremendously grateful to everyone who has submitted a pull request on GitHub, corrected typos in the our content, expressed encouragement on the Coursera forums or by email, or engaged with us in any other way. There are far too many of these people to name, but it is safe to say that swirl would be far worse off without them.

Finally, I would like to thank my advisers from the Johns Hopkins Department of Biostatistics, Jeff Leek, Brian Caffo, and Roger Peng. I could go on for many pages about everything they have done

for me. They have provided indispensable advice and encouragement since I first started working on swirl. In fact, the idea for swirl came out of a conversation that Brian and I had in his office during summer 2013. What I am most grateful for, though, is the confidence that they have instilled in me to think big, to trust my intuition, and to forge ahead in spite of the naysayers.

3 Introduction

3.1 A very brief introduction to interactive, computer-based learning

The idea of using technology to create an interactive learning environment has been around a long time. Sidney Pressey developed the first teaching machines in the 1920s, long before the advent of modern computers (Benjamin 1988). B.F. Skinner later popularized teaching machines in the 1950s, coining the term *programmed instruction* to describe his method of introducing new material to students in very small steps (Skinner 1954; Skinner 1958; Skinner 1986). Both Pressey's and Skinner's machines followed a similar flow: present a question to the student, solicit a response, evaluate the response, give immediate feedback, then proceed to the next question. In contrast to the traditional classroom model, students using these machines learned entirely at their own pace.

In the 1960s, researchers at the University of Illinois developed PLATO (Programmed Logic for Automatic Teaching Operations), the first generalized computer assisted instruction system (Wikipedia 2014a). The TUTOR programming language was developed as a framework for allowing anyone to author lessons for PLATO. As systems for computer-based learning became increasingly sophisticated, the field of intelligent tutoring systems (ITSs) emerged in the 1980s (Yazdani 1986). The domain of computer programming was a good fit for early experimentation with ITSs and two platforms had emerged by the mid-1980s: one for the LISP programming language (Anderson and Reiser 1985) and the other for Pascal (Johnson and Soloway 1985).

Widespread access to the internet and modern personal computers changed the face of computer-based learning over the years. Today's learners have access to Massive Open Online Courses (MOOCs), distance learning programs, online video tutorials, interactive textbooks, interactive programming sites, and more. Despite its many forms, one thing has become clear: computer-based instruction, when implemented thoughtfully, can significantly improve learning outcomes for students (Kulik and Kulik 1991; Barrow, Markman, and Rouse 2008; Sosa et al. 2011; Tamim et al. 2011).

In recent years, interactive platforms for learning computer programming have exploded in popularity. Web applications like Codecademy (Codecademy 2014), Code School (Code School LLC 2014), and Treehouse (Treehouse Island Inc. 2014) have become household names. Some platforms are free and others are paid, but they are all interactive in the sense that a user actively solves problems and receives real-time feedback as they progress through increasingly difficult lessons in a quest to improve their programming skills.

With only a couple of exceptions (see below), these tools have focused on mainstream languages such as Python, Ruby, JavaScript, HTML, and CSS. Until very recently, the R programming language (R Core Team 2014) has been conspicuously absent from the world of interactive, computer-based learning.

3.2 Resources for learning R interactively

Aside from swirl, we are aware of two resources for learning R interactively, both of which are web applications.

3.2.1 Try R

Try R (Code School 2014), created by Code School and sponsored by O'Reilly, provides a series of eight online lessons covering basic topics in R programming. Content authoring does not appear to be open to the public. Lessons, which must be completed in order, lead users through a series of questions, moving them linearly down the page as they progress. It seems that the application is not under active development.

3.2.2 DataCamp

DataCamp (DataCamp 2014a), also a web application, currently offers four built-in courses varying in difficulty and covering topics in R programming, statistics, and data analysis. Content authoring is open to the public and instructors can create and upload courses using the datacamp and datacampSCT R packages (DataCamp 2014b).

DataCamp's interface, which is clean and easy to use, will be familiar to anyone who has used Codecademy. For most questions in DataCamp, the screen is split into three parts: an instruction pane, a script, and a console. The platform also offers multiple choice questions, which conveniently allow the user to play around with code in a console window before selecting an answer. The platform is under active development and new courses are forthcoming.

3.3 Motivation behind swirl choices

No platform is perfect and user preferences vary. In what follows, we point out some of the differences between swirl and the other platforms.

3.3.1 Learn R, in R

The most obvious difference between swirl and existing platforms for learning R interactively is that swirl is an R package that runs directly in the R console. While the R console is not (and never will be) as aesthetically pleasing as a carefully crafted web application, it is where real data analysis takes place. For this reason, we hang our hat on the authenticity of the learning environment that swirl provides. There is no separation between where users of swirl learn R and where they conduct data analysis when they are not using swirl.

3.3.2 Continuous session, like real data analysis

A swirl lesson is a continuous dialog between swirl and the user, taking place primarily in the R console. Because of this, the session is cumulative and continuous, just like a real data analysis. In other words, the entire history of the session is recorded in the console in a linear fashion and any variables created during the session persist until the user clears her workspace.

Try R has a similarly linear flow to it, but the environment is artificial. DataCamp takes a different approach, such that lessons are a series of discrete mini-projects, each of which begins with a fresh set of instructions, an R script, and an empty console.

3.3.3 Runs locally

Once a user has installed a course, swirl does not require an internet connection to run. This can be particularly helpful for users who may not have access to a consistent, reliable internet connection. While we are actively exploring ways in which we can incorporate useful web technologies into swirl, we intend for those features to either run locally, or be optional to the user, so that swirl may always be run entirely offline if desired.

3.3.4 100 percent open source

In the spirit of R, all software and instructional content associated with swirl are totally open source and available on GitHub (GitHub, Inc. 2014) under the *swirldev* organization (swirl Development Team 2014a). We believe this level of transparency is a significant advantage to learners, instructors, and contributors.

3.3.5 Anyone can author and share content

Anyone can author interactive swirl content using the swirlify R package (swirl Development Team 2014b), which offers a convenient set of tools to make the process simple and straightforward. swirl users can install courses directly from a variety of sources, including GitHub, Dropbox, Google Drive, an arbitrary URL, or a local directory or zip file. This makes it fast and easy for content authors to share their creations either publicly or privately. While Try R does not allow others to contribute content, DataCamp seems to have a well-developed system for doing so.

3.3.6 Integration with Coursera API

Coursera is a popular platform for Massive Open Online Courses (MOOCs)(Coursera 2014a). swirl offers direct integration with Coursera's gradebook API (Coursera 2014b). To date, swirl has been used as the interactive component for three courses, all part of the Johns Hopkins Data Science Specialization (JHU Data Science 2014). Students are able to have swirl automatically notify Coursera of each lesson they complete, upon which credit is granted on the Coursera website. This has proven to be a very useful feature for learners and we hope to make this model more widely available to instructors in more conventional classroom settings.

3.4 Measuring swirl's impact

Because swirl runs locally and does not collect data on users, measuring its impact is somewhat of a challenge. swirl is available for download from CRAN (Comprehensive R Archive Network), which is a collection of sites carrying, among other things, contributed R packages (Hornik 2014). These sites, called *mirrors*, are not required to publish their download data, but fortunately the mirror hosted by RStudio does make this data publicly available (RStudio 2014a).

swirl was made first made available on CRAN on January 14, 2014. As of September 8, 2014, it has been downloaded 72,819 times from the RStudio CRAN mirror. There are 100 CRAN mirrors worldwide (according to `chooseCRANmirror()`), so this number is probably a significant underestimate of total downloads.

We also maintain a website, swirlstats.com. Since January 14, there have been 62,911 unique visitors to the website and 227,668 pageviews. We have had visitors from every country in the world. The source code for swirl and all of our courses are hosted on GitHub. As of September 8, the course

repository (swirl Development Team 2014c) has been forked 594 times and starred 322 times. The swirl repository (swirl Development Team 2014d) has been forked 74 times and starred 137 times.

4 Installing swirl

The first step to using swirl for teaching or learning is to install the package.

4.1 From CRAN

The latest stable release of swirl is most easily installed from CRAN.

```
R> install.packages("swirl")
```

4.2 From GitHub

The most recent development version of swirl is most easily installed from GitHub using the devtools R package.

```
R> install.packages("devtools")
R> devtools::install_github("swirldev/swirl")
```

5 A typical swirl session

Before discussing swirl in detail, it is helpful to see how a typical swirl session looks and feels to the user.

5.1 Loading swirl

```
R> library(swirl)
```

```
| Hi! Type swirl() when you are ready to begin.
```

5.2 Starting swirl

```
R> swirl()

| Welcome to swirl!

| Please sign in. If you've been here before, use the same name as
| you did then. If you are new, call yourself something unique.

What shall I call you? Nick

| Thanks, Nick. Let's cover a few quick housekeeping items before we
| begin our first lesson. First of all, you should know that when you
| see '...', that means you should press Enter when you are done
| reading and ready to continue.

...  <-- That's your cue to press Enter to continue

| Also, when you see 'ANSWER:', the R prompt (>), or when you are
| asked to select from a list, that means it's your turn to enter a
| response, then press Enter to continue.

Select 1, 2, or 3 and press Enter

1: Continue.
2: Proceed.
3: Let's get going!

Selection: 2

| You can exit swirl and return to the R prompt (>) at any time by
| pressing the Esc key. If you are already at the prompt, type bye()
| to exit and save your progress. When you exit properly, you'll see
```

```
| a short message letting you know you've done so.

| When you are at the R prompt (>):
| -- Typing skip() allows you to skip the current question.
| -- Typing play() lets you experiment with R on your own; swirl will
| ignore what you do...
| -- UNTIL you type nxt() which will regain swirl's attention.
| -- Typing bye() causes swirl to exit. Your progress will be saved.
| -- Typing main() returns you to swirl's main menu.
| -- Typing info() displays these options again.

| Let's get started!
```

5.3 Installing a course

```
| To begin, you must install a course. I can install a course for you
| from the internet, or I can send you to a web page
| (https://github.com/swirldev/swirl_courses) which will provide
| course options and directions for installing courses yourself. (If
| you are not connected to the internet, type 0 to exit.)

1: R Programming: The basics of programming in R
2: Regression Models: The basics of regression modeling in R
3: Don't install anything for me. I'll do it myself.

Selection: 1

| Course installed successfully!
```

5.4 Selecting a course and lesson

```
| Please choose a course, or type 0 to exit swirl.
```

```
1: R Programming
2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

 1: Basic Building Blocks      2: Sequences of Numbers
 3: Vectors                    4: Missing Values
 5: Subsetting Vectors         6: Matrices and Data Frames
 7: lapply and sapply          8: vapply and tapply
 9: Looking at Data           10: Simulation
11: Dates and Times

Selection: 1

  |                                                              |   0%

| In this lesson, we will explore some basic building blocks of the R
| programming language.

...
```

5.5 Continuous dialog between swirl and the user

A swirl lesson is a continuous session that takes place primarily in the R console. Control alternates between swirl (i.e. instructions) and the user (i.e. responses).

```
  |==========                                                  |  19%

| To assign the result of 5 + 7 to a new variable called x, you type
| x <- 5 + 7. This can be read as 'x gets 5 plus 7'. Give it a try
| now.
```

```
R> x <- 5 + 7

| All that practice is paying off!

  |============                                                  |  22%

| You'll notice that R did not print the result of 12 this time. When
| you use the assignment operator, R assumes that you don't want to
| see the result immediately, but rather that you intend to use the
| result for something else later on.

...

  |===============                                               |  24%

| To view the contents of the variable x, just type x and press
| Enter. Try it now.

R> x
[1] 12

| You nailed it! Good job!
```

5.6 In-lesson options

Anytime the user is at the R prompt during a lesson, he can type `info()` for a menu of options.

```
  |================================                              |  54%

| Take the square root of z - 1 and assign it to a new variable called
| my_sqrt.
```

```
R> info()

| When you are at the R prompt (>):
| -- Typing skip() allows you to skip the current question.
| -- Typing play() lets you experiment with R on your own; swirl will ignore what you do...
| -- UNTIL you type nxt() which will regain swirl's attention.
| -- Typing bye() causes swirl to exit. Your progress will be saved.
| -- Typing main() returns you to swirl's main menu.
| -- Typing info() displays these options again.
```

5.7 Getting things wrong

One of the primary benefits of an interactive learning environment is that the user receives immediate feedback upon entering an incorrect response.

```
  |================================                      |  54%

| Take the square root of z - 1 and assign it to a new variable called
| my_sqrt.

R> my_sqrt <- sqrt(z)

| That's not exactly what I'm looking for. Try again. Or, type info()
| for more options.

| Assign the result of sqrt(z - 1) to a variable called my_sqrt.

R> my_sqrt <- sqrt(z - 1)

| Great job!
```

5.8 Various question types

swirl supports many question types, all of which will be discussed in more detail later. One common and useful question type is multiple choice.

```
  |==================================                    |  57%

| Before we view the contents of the my_sqrt variable, what do you
| think it contains?

1: a vector of length 0 (i.e. an empty vector)
2: a single number (i.e a vector of length 1)
3: a vector of length 3

Selection: 3

| All that hard work is paying off!
```

5.9 Exiting swirl

```
  |=======================================               |  65%

| Now, create a new variable called my_div that gets the value of z
| divided by my_sqrt.

R> bye()

| Leaving swirl now. Type swirl() to resume.
```

5.10 Resuming progress

```
R> swirl()

| Welcome to swirl!
```

```
| Please sign in. If you've been here before, use the same name as you
| did then. If you are new, call yourself something unique.

What shall I call you? Nick

| Would you like to continue with one of these lessons?

1: R Programming Basic Building Blocks
2: No. Let me start something new.

Selection: 1

| Now, create a new variable called my_div that gets the value of z
| divided by my_sqrt.

R>
```

5.11 Notifying Coursera

Many users first experience swirl as part of a Coursera course. These users have the option to automatically receive credit upon completing swirl lessons.

```
| Are you currently enrolled in the Coursera course associated with
| this lesson?

1: Yes
2: No

Selection: 1

| Would you like me to notify Coursera that you've completed this
| lesson? If so, I'll need to get some more info from you.
```

```
1: Yes
2: No
3: Maybe later

Selection: 1

| The first item I need is your Course ID. For example, if the homepage
| for your Coursera course was 'https://class.coursera.org/rprog-001',
| then your course ID would be 'rprog-001' (without the quotes).

Course ID: rprog-007
Submission login (email): nick@example.com
Submission password: ********

| Is the following information correct?

Course ID: rprog-007
Submission login (email): nick@example.com
Submission password: ********

1: Yes, go ahead!
2: No, I need to change something.

Selection: 1

| I'll try to tell Coursera you've completed this lesson now.

| Great work!

| I've notified Coursera that you have completed rprog-007,
| Basic_Building_Blocks.
```

```
| You've reached the end of this lesson! Returning to the main menu...
```

6 How swirl works

6.1 Parsing a swirl lesson

Authoring swirl lessons will be discussed in detail later in the paper. For now, it is sufficient to know that lessons are authored using a simple `key:value` syntax called YAML (Evans 2014). When the user selects a lesson from the lesson menu, swirl locates the lesson, parses the YAML using the `yaml.load_file` function from the yaml R package, then converts the resulting list to a data frame.

Each row of the data frame contains exactly one 'unit' of content (e.g. one block of text output, one multiple choice question, etc.). Each column of the data frame represents a specific type of information (e.g. what text to output to the user, the correct answer to a question, the name of a file containing a figure to be displayed, etc.)

6.2 Lesson provides instructions

Once the lesson has been converted to a data frame, swirl steps through the data frame one row at a time. Each row contains specific instructions for how swirl should behave. For example, should swirl display some text output, then wait for the user to press Enter? Should it ask a question, then wait for the user to enter a valid R command at the prompt? If so, what response from the user should be deemed correct? What should swirl do if it's incorrect?

6.3 Callback mechanism

The first iteration of swirl made heavy use of the `readline` function in base R to solicit responses from the user. In this version of the software, when the user entered an R command in response to a question, he was not interacting directly with the R interpreter. We used the `>` character to imitate the look of the R prompt, but the input was actually read in as character string instead of an R expression. We were still able to use R's `parse()` function to parse the input in the background, but the user lost out on receiving authentic feedback from the R interpreter when, for example, he entered an invalid R expression that would have normally caused an error message to be displayed.

The `addTaskCallback` function was first recommended to us by Hadley Wickham. He demonstrated it's use with frndly.R (Wickham 2013). Using the task callback mechanism invoked by `addTaskCallback`, we could now allow the user to interact directly with the R interpreter, while swirl "listened" in the background. We simply captured the important aspects of the user's response and had swirl react accordingly.

The `swirl` function shown below is really the workhorse of the entire package.

```
swirl <- function(resume.class="default", ...){
  removeTaskCallback("mini")
  e <- new.env(globalenv())
  class(e) <- c("environment", resume.class)
  cb <- function(expr, val, ok, vis, data=e){
    e$expr <- expr
    e$val <- val
    e$ok <- ok
    e$vis <- vis
    return(resume(e, ...))
  }
  addTaskCallback(cb, name="mini")
  invisible()
}
```

Because it is so important, let's break it down into smaller pieces. The first thing that happens when the user starts swirl is that any existing callbacks leftover from previous swirl sessions are removed with `removeTaskCallback("mini")`, where `mini` is the name we assign (later in the function) to the callback.

Next, we create a new environment `e`, which is essentially a `key:value` structure which can also act as a context for executing functions, and attach a second `class` to it based on the first (optional) argument to `swirl`. `swirl` is intended to be called by the user with no arguments, but by altering the value of `resume.class`, we are able to manipulate swirl's behavior by making use of R's S3 system for object oriented programming. The details of this are beyond the scope of this paper, but suffice it to say that this has helped us run swirl in special "test" modes intended for testing content or debugging the software.

`addTaskCallback(cb, name="mini")` initializes a callback named `mini`, instructing R to pass a particular set of information as arguments to the `cb` function defined just above it. The way `addTaskCallback` works is that every time an expression is entered at the prompt (a so-called top-level task), it passes four arguments to the function specified (`cb` in this case). As defined in the function documentation, the arguments are (in order) "the expression for the top-level task, the result of the top-level task, a logical value indicating whether it was successfully completed or not (always TRUE at present), and a logical value indicating whether the result was printed or not." The fifth argument to `cb` allows us to use the environment `e` within the function.

The first two arguments to `cb`, which we label `expr` and `val` are most important. `expr` is a parsed, but unevaluated, expression, representation of the expression entered by the user. `val` is the value returned when the expression is evaluated. For example, if the user entered `x <- 5 + 7` at the prompt, `expr` would contain the unevaluated R expression `x <- 5 + 7` and `val` would contain the numeric value `12`.

Here's where the magic happens. `cb` saves copies of its first four arguments to the environment `e`, then returns the result of passing `e` to the `resume` function. This starts a series of events, which ultimately results in swirl reacting to the user's response appropriately, then returning `TRUE` if the callback should remain alive, or `FALSE` if it should be removed. You might think of the callback as being being swirl's heartbeat, and when it is removed, the swirl program ends.

The final line of the `swirl` function, `invisible()`, just prevents anything from printing to the R console when the function has completed execution.

7 Authoring swirl content

From the start, we wanted swirl to be something that the R and statistics communities could build on and embrace as their own. We have created several swirl courses and made them available for free to everyone. However, the tools we used to build this content are also freely available and we are aware of many instructors around the world who have created swirl courses of their own. Our vision is that swirl will provide an effective, easy-to-use, and flexible platform for others to create and share interactive courses.

7.1 Existing content

The remainder of this section will outline the steps required to create custom swirl content and share it with others. We have created a great deal of instructional content ourselves and recommend using it as a guide for creating new content.

All of the courses that we have created are contained in our course repository on GitHub (swirl Development Team 2014c). As of September 8, 2014, we have seven courses available, although some are only partially completed. Our current offerings are as follows:

- **R Programming**: The basics of programming in R
  - Basic Building Blocks
  - Sequences of Numbers
  - Vectors
  - Missing Values
  - Subsetting Vectors
  - Matrices and Data Frames
  - lapply and sapply
  - vapply and tapply
  - Looking at Data
  - Simulation
  - Dates and Times
- **R Programming Alt**: Same as the original, but modified slightly for in-class use (see below*)
- **Data Analysis**: Basic ideas in statistics and data visualization
  - Central Tendency
  - Dispersion
  - Data Visualization
- **Mathematical Biostatistics Boot Camp**: One- and two-sample t+tests, power, and sample size
  - One Sample t-test
  - Two Sample t-test
  - Errors, Power, and Sample_Size

- **Open Intro**: A very basic introduction to statistics, data analysis, and data visualization
  - Overview of Statistics
- **Regression Models**: The basics of regression modeling in R
  - Introduction
  - Residuals
  - Least Squares Estimation
  - Residual Variation
  - Introduction to Multivariable Regression
  - Multivariable Examples
  - Multivariable Examples 2
  - Multivariable Examples 3
  - Residuals Diagnostics and Variation
  - Variance Inflation Factors
  - Overfitting and Underfitting
  - Binary Outcomes
  - Count Outcomes
- **Getting and Cleaning Data**: dplyr, tidyr, lubridate, oh my!
  - Manipulating Data with dplyr
  - Grouping and Chaining with dplyr
  - Tidying Data with tidyr
  - Dates and Times with lubridate

* R Programming Alt is identical to R Programming, except we eliminated the prompts for Coursera credentials at the end of each lesson and instead give students the option to send an email to their instructor notifying them of completion. This is a temporary solution for in-class use until we develop something more robust (i.e. instructor dashboard).

7.2 Authoring tools (swirlify)

We have collected a set of tools that streamline the creation of interactive swirl content. These tools are available in an R package called swirlify, which is available on GitHub (swirl Development Team 2014b).

7.3 Installing swirlify

The easiest way to install swirlify is with the devtools package. The development version of swirl should be installed with it, to ensure access to the most up-to-date features.

```
R> install.packages("devtools") # If not already installed
R> devtools::install_github(c("swirldev/swirl", "swirldev/swirlify"))
```

7.4 Creating a new lesson

swirl courses are collections of individual lessons. The following code will load swirlify and create a new lesson. If swirl does not find a course directory that matches the course name provided, a new course will be created as well.

```
new_lesson("My Lesson", "My Course")
```

7.5 Working on an existing lesson

You can set the lesson that you would like to work on either interactively using a file menu, or by specifying the full path to the lesson file you wish to work on.

```
set_lesson() # Set the lesson interactively
## OR...
set_lesson("path/to/My_Course/My_Lesson/lesson.yaml") # Set directly
```

7.6 Anatomy of a swirl course

Here is an example of how a swirl course directory structure might look.

```
|- My_Course
   |- A_Lesson
      |- lesson.yaml
      |- initLesson.R
      |- customTests.R
      |- scripts
```

```
   |- some-data.csv
   |- dependson.txt
|- Another_Lesson
   |- lesson.yaml
   |- initLesson.R
   |- customTests.R
|- One_More_Lesson
   |- lesson.yaml
   |- initLesson.R
   |- customTests.R
   |- more-data.Rda
|- MANIFEST
```

7.6.1 Course directory

The course directory contains all information relevant to an individual swirl course. Within it, each lesson has its own directory. There is also a file called `MANIFEST`, within which the instructor lists the names of the lesson directories (one per line) in the order in which the lessons should be displayed to the user in the lesson menu.

7.6.2 Lesson directory

Each lesson directory contains all information relevant to a particular lesson.

7.6.2.1 dependson.txt This file allows the instructor to list (one per line) the names of any R packages that the lesson requires. When a user selects the lesson, swirl will attempt to install (from CRAN) and load each package listed in this file.

7.6.2.2 initLesson.R swirl will `source` this file in the global environment at the start of the lesson. This allows the instructor to predefine variables for the user or perhaps load a dataset into the user's workspace.

7.6.2.3 Data Any data to be made available to the user during a swirl lesson may be included in the lesson directory. courses are installed in the package directory, which is not ideal, but the

rationale will be discussed at length later in the paper. Because of this arrangement, it is easiest for the instructor to create a variable that contains the full path to data and either make that available in the user's workspace or go a step further and load the data for the user. The former involves using `find.package` or `system.file` to locate the installed swirl package on the user's computer, then using knowledge of the course directory structure to locate the data file.

For example, the following would create a variable in the user's workspace called `path2data` that contains the full file path to some-data.csv from our example course outlined above.

```
path2data <- file.path(find.package("swirl"), "Courses",
                       "My_Course", "A_Lesson", "some-data.csv")
```

The latter requires a call to `data`, `read.csv`, or the like. For example, the following code would create a variable called `dat` in the user's workspace that contains the data required for the lesson.

```
dat <- read.csv(path2data, stringsAsFactors = FALSE)
```

7.6.2.4 lesson.yaml This is the most important (and actually the only *required*) item in the lesson directory. It contains all lesson content and follows a particular format discussed in more detail below.

7.6.2.5 customTests.R Answer tests will also be discussed in more detail later in the paper. For now, just know that swirl uses logic provided by the instructor to determine if any given response correctly answers a question posed to the user. Besides having a library of predefined answer tests to draw from, instructors may define their own custom answer tests in a file named customTests.R to satisfy their particular needs.

7.6.2.6 script directory This directory contains R scripts for script type questions. Script questions present an R script to the user for her to edit, then submit when she is ready. Each question has two scripts associated with it – one to be presented to the user and the other to represent one possible solution to the question. These will be addressed in more detail below.

7.6.2.7 Other Instructors may wish to store other files containing notes or outlines in the lesson directory. This is fine and will have no effect on swirl's behavior when running the lesson. However,

any files included in the lesson directory will be installed on the users computer along with the rest of the course, so discretion should be used.

7.7 Content format

In the early days of swirl, content authoring was done directly in comma-separated value (CSV) files. It seemed most logical to organize swirl lessons into tabular format, so authoring in CSV files was the most direct approach to authoring. The lesson data were well organized and no intermediate parsing was necessary. Unfortunately, we found it to be very unnatural to write lessons in tabular format. We sought a more linear approach that was more amenable to long-form writing.

In an attempt to capitalize on recent trends, we created a swirl flavor of R Markdown (RStudio 2014b). However, this approach also had limitations, the most significant of which was lack of structure. R Markdown is certainly more natural for authoring long-form content, but the data is inherently unstructured.

We sought a compromise with YAML, which is very human readable, but also highly structured. YAML is familiar to many people, but even for content authors who are new to YAML, it is easily grasped. Our most recent generation of authoring tools have been constructed around the YAML approach and, based on our own experience and feedback from others, it seems to be an effective method for authoring swirl content.

7.8 Units of instruction

Courses consist of lessons, which in turn consist of units of instruction. There are many different unit types. In what follows, we provide a description of each, followed by an example of that unit type.

7.8.1 Text output

This is the most basic unit of instruction and merely involves outputting some text to the user. After the text, swirl displays `...` and waits for the user to press Enter before continuing on to the next unit.

```
- Class: text
  Output: "Now, suppose we want to simulate 100 flips of an
```

```
    unfair two-sided coin. This particular coin has a 0.3
    probability of landing 'tails' and a 0.7 probability of
    landing 'heads'."
```

7.8.2 Command questions

A command question outputs some instructions, then returns the user to the R prompt. If the user enters a valid R expression at the prompt which is judged by swirl to be correct, then the user advances to the next unit. If the user's response is incorrect, a hint is displayed and the user is returned to the prompt for another attempt.

```
- Class: cmd_question
  Output: "Now that we have a sense of the shape and size of the
    dataset, let's get a feel for what's inside. names(plants)
    will return a character vector of column (i.e. variable)
    names. Give it a shot."
  CorrectAnswer: names(plants)
  AnswerTests: omnitest(correctExpr='names(plants)')
  Hint: "names(plants) will return a character vector of column
    (i.e. variable) names."
```

7.8.3 Script questions

This is the newest type of unit. Text instructions are displayed in the console and the specified R script is opened automatically for the user. The user may edit and save it, then type `submit()` when ready to move on. If the answer is deemed correct, the user advances to the next unit. If not, then a hint is displayed and the user is allowed another attempt. The user may also type `reset()` to discard any changes and return the script to it's original state.

```
- Class: script
  Output: "Now, using the script I just opened for you, create a
    second table called gradebook using the id, class, midterm,
    and final columns (in that order).\n\nEdit the R script and
    type submit() when you are ready. Type reset() to reset the
```

```
    script to its original state."
  AnswerTests: script_vals_identical()
  Hint: "Read the directions in the script carefully. If R is
    giving you an error, try to understand what it is telling
    you. Type submit() at the prompt when you are ready, or type
    reset() to reset the script to its original state."
  Script: script7.R
```

7.8.4 Multiple choice questions

Multiple choice questions display a question in the console, then present the user with two or more possible correct answers. The user may select one at a time and if correct, will advance to the next unit. If incorrect, a hint is displayed and the question is repeated.

```
- Class: mult_question
  Output: What do you think x[is.na(x)] will give you?
  AnswerChoices: "A vector of all NAs; A vector with no NAs;
    A vector of TRUEs and FALSEs; A vector of length 0"
  CorrectAnswer: A vector of all NAs
  AnswerTests: omnitest(correctVal="A vector of all NAs")
  Hint: "Remember that is.na(x) tells us where the NAs are in a
    vector. So if we subset x based on that, what do you expect
    to happen?"
```

7.8.5 Figures and plots

Figure units allow the instructor to display a plot or other figure to the user by sourcing a specified script containing the code necessary to generate the figure. In reality, this script could contain any code the instructor wished to execute for the user, but it's originally intended purpose was for generating figures and plots.

The `FigureType` field tells swirl whether the figure is a `new` plot or `add`s some feature to an existing plot. This only matters when the user exits swirl, then later restores progress. If a script assumes that a previously created plot is present in the plotting window, when in fact it is not, this will cause

an error. A classic example would be a `new` plot generated using the `plot` command and an `add` plot that adds a regression line to the existing plot with the `abline` command.

```
- Class: figure
  Output: "Here is a plot of Galton's data, a set of 928
    parent/child height pairs. Moms' and dads' heights were
    averaged together (after moms' heights were adjusted by a
    factor of 1.08). In our plot we used the R function "jitter"
    on the children's heights to highlight heights that occurred
    most frequently. The dark spots in each column rise from
    left to right suggesting that children's heights do depend
    on their parents'. Tall parents have tall children and short
    parents have short children."
  Figure: plot1_children_vs_parents.R
  FigureType: new

- Class: figure
  Output: "Here we add a red (45 degree) line of slope 1 and
    intercept 0 to the plot. If children tended to be the same
    height as their parents, we would expect the data to vary
    evenly about this line. We see this isn't the case. On the
    left half of the plot we see a concentration of heights
    above the line, and on the right half we see the
    concentration below the line."
  Figure: plot2_identity_line.R
  FigureType: add
```

7.8.6 Videos and URLs

Video units ask the user if they would like to view a video on a particular topic. If yes, then swirl attempts to open in the user's default web browser a URL provided by the instructor. Actually, this unit type could be used to prompt the user to open any URL, but it's original intent was for incorporating into lessons supplemental video content from the web.

```
- Class: video
  Output: "Would you like to watch a brief YouTube video on
    stem-and-leaf plots?"
  VideoLink: http://youtu.be/0aJXJduRiIE
```

7.9 Sharing content

swirl is designed so that anyone can create interactive content and share it with the world or with just a few people. Users can install (and uninstall) courses from a variety of sources using the functions listed here. Each of these functions has its own help file, which may be consulted for more details.

- `install_from_swirl`: Install from the official course repository
- `install_course_github`: Install from a GitHub repository
- `install_course_google_drive`: Install from Google Drive
- `install_course_dropbox`: Install from Dropbox
- `install_course_url`: Install from an arbitrary URL
- `install_course_zip`: Install from a local zip file
- `install_course_directory`: Install from local directory
- `uninstall_course`: Uninstall a course

8 Answer testing

Answer tests are how swirl determines whether a user has answered a question correctly or not. Each question has one or more answer tests associated with it, all of which must be satisfied in order for a user's response to be considered correct. An instructor can specify any combination of our predefined answer tests or create her own custom answer tests to suit her specific needs.

For each question an instructor authors as part of a swirl lesson, she must specify exactly one correct answer. This is separate and distinct from the answer tests. This does not have to be the only correct answer, but it must answer the question correctly. If a user skips the question, this is the answer that will be entered on his behalf.

Some question types simply need to check that the user entered the correct value. For example, a multiple choice question typically has exactly one correct answer. Questions that require the user to

enter a valid command at the R prompt (i.e. command questions) are more complicated. Since there are often many possible ways to answer a command question, the instructor must determine how she would like swirl to assess the correctness of a user's response. This is where answer tests come in.

Instructors can specify any number of answer tests. If more than one test is used, they must be separated by semicolons. If an instructor does not wish to specify more specific answer tests for a command question, then the default test can be used. The default test is `omnitest(correctExpr='<correct_answer_here>')`, which will simply check that the user's expression matches the expression that the instructor provided as a correct answer.

In many cases, the default answer test will provide sufficient vetting of a user's response to a command question. While it is somewhat restrictive in the sense that it requires an exact match of expressions (ignoring whitespace), it is liberating to the course author for two reasons.

First, it allows for fast prototyping of content. As an instructor is developing content, she may find that determining how to test for correctness distracts her from the message she is trying to communicate.

Second, the instructor does not have to worry about what happens if the user enters an incorrect response, but is allowed to proceed because of an oversight in the answer tests. Since swirl sessions are continuous, accepting an incorrect answer early in a lesson can cause problems later on. By using the default answer test, an instructor eliminates this burden by requiring an exact match of expressions and hence not allowing the user to advance until she is certain they have entered the correct response.

Instructors can always go back later on and make their answer testing strategies more elaborate. The main benefit of using tests other than the default is that the user will not be required to enter an expression exactly the way the instructor has specified it. He or she will have more freedom in terms of how they respond to a question, as long as they satisfy the conditions that the instructor sees as being most important.

8.1 Predefined Answer Tests

Each of the predefined answer tests listed below has its own help file, where you can find more detailed explanations and examples.

- `any_of_exprs`: Test that the user's expression matches any of several possible expressions.

- `expr_creates_var`: Test that a new variable has been created.
- `expr_identical_to`: Test that the user has entered a particular expression.
- `expr_is_a`: Test that the expression itself is of a specific class (e.g. is an assignment).
- `expr_uses_func`: Test that a particular function has been used.
- `func_of_newvar_equals`: Test the result of a computation such as `mean(newVar)` applied to a specific (user-named) variable created in a previous question.
- `omnitest`: Test for a correct expression, a correct value, or both.
- `val_has_length`: Test that the value of the expression has a particular length.
- `val_matches`: Test that the user's expression matches a regular expression (`?regex`).
- `var_is_a`: Test that the value of the expression is of a specific class.

8.2 Custom Answer Tests

Occasionally, instructors may want to test something that is outside the scope of our predefined answer tests. Fortunately, this is very easy to do. If an instructor is using the swirlify package to author content, then a file called customTests.R (case-sensitive) is automatically created in the lesson directory. If it's not there already, he can create the file manually.

In this file, an instructor can write his own answer tests. These answer tests are then available to him just the same as any of the standard tests. However, the scope of a custom answer test is limited to the lesson within which it is defined.

Each custom answer test is simply an R function that follows a few basic rules:

- Has a distinct name that will help the instructor remember what is does (e.g. `creates_matrix_with_n_rows`).
- The first line of the function body is `e <- get("e", parent.frame())`, which gives the function access to the environment `e`. Any important information, such as the expression typed by the user, will be available through `e`.
- Accesses the expression entered by the user with `e$expr` and the value of the expression with `e$val`. Note that `e$expr` comes in the form of an unevaluated R expression.
- The function returns `TRUE` if the test is passed and `FALSE` otherwise. The instructor should be careful that no other value could be returned (e.g. `NA`, `NULL`, etc.)

9 The future of swirl

swirl has evolved rapidly since it was created in summer 2013, but we still have long way to go. This section will outline some of the changes we would like to make over the coming years to make swirl a more valuable tool for learning and teaching R and statistics.

9.1 Design issues

When we first started writing code for swirl, we had no idea what it was going to look like. Now that swirl has taken shape and its role has become clear, there are many design issues that we would like to address.

9.1.1 Performance

One of the nice features of swirl is that as a user is working through a lesson, his progress is saved after each unit of instruction. Essentially, this means that a copy of the user's workspace is made at each step along the way. Thus, regardless of whether the user exits intentionally (with the `bye` command) or accidentally, progress is retained and may be restored upon restarting swirl.

Unfortunately, when we implemented this feature, we were not anticipating the implications for when an instructor used a large dataset in a lesson. Loading a large amount of data meant that this data would be saved repeatedly throughout the lesson, causing things to run very slowly.

To address the problem, we created a flag called `AUTO_DETECT_NEWVAR`, which could be controlled by the instructor in the customTests.R file. When this flag was set to `FALSE`, swirl no longer would automatically detect the creation of a new variable (e.g. containing a large dataset) and would therefore not automatically save a copy of it. The instructor could then handpick which variables the user creates during a lesson that should be retained in case the user exits and later resumes progress.

The upside of this approach is that the instructor can have swirl avoid repeatedly saving copies of a large dataset. The downside is that if the user exits after loading a dataset that hasn't been saved to progress, then later resumes, the lesson will not restore the dataset to the workspace and therefore the lesson will not operate correctly if the dataset is called upon later on.

The solution to this dilemma is not immediately clear. It is possible that a copy of the global environment could be saved only when the global environment is deemed to have changed in some

way. Other progress markers, such as how far along in the lesson someone is, could still be updated after each unit of content.

Another alternative would be to not save a snapshot of the global environment at all. When the user quit swirl, then resumed progress partway through a lesson, swirl could simply evaluate (in order) the "correct answers" that the instructor specified for each of the questions prior to the current unit. This would significantly improve performance, since swirl would not attempt to make repeatedly make copies of objects. The greatest downside would manifest itself when "correct answers" were not deterministic.

In other words, if there was some random element, such a random number generated from `rnorm` or `sample`, or if the user was allowed to use some discretion in, say, the name or value of a variable. When progress was restored, the values generated from evaluating the "correct answers" specified by the instructor would almost certainly not match those previously generated by the user.

9.1.2 Storing courses and data in home directory

There has been an ongoing discussion among swirl's developers as to the most appropriate place to store data such as installed courses, user progress, and other user information, on the user's computer. To date, our solution has been to save everything to the installed package directory. The rationale behind this was that we figured it was the least invasive option, given that there was not really a precedent for this among other R packages. This method has not been without problems, most notable of which is that many users (particularly those on work or school computers) have restricted file access and aren't able to write to the swirl package directory.

We've discussed for some time now the more desirable approach of having swirl write data to the user's home directory, but this, too, is not totally straightforward. The path to a user's home directory is very platform-dependent.

One elegant solution to this problem is posed by the rappdirs R package, created by Hadley Wickham, which is an R version of the popular appdirs (Active State 2014) software originally written in Python. rappdirs (Wickham 2014b) solves the problem of finding the user's home directory based on her operating system. Until very recently, rappdirs was only available on GitHub, but is now on CRAN. We anticipate making full use of it in the near future to shift all course and user data to a suitable subdirectory of the user's home directory.

9.1.3 Course anatomy

When we first started making swirl courses, we put everything in CSV files. As swirl grew in complexity and capability, we needed to accommodate more metadata and other accompanying files. Things evolved naturally into their current state, where each course has its own directory, which in turn contains multiple lesson directories, each of which has several files including the main lesson content, any data files, initialization code, custom tests, etc. The course structure we've ended up with, while effective, has no precedent among R packages. As a result, we've had to invent our own functionality for many things, such as reading lesson metadata from the top of a lesson YAML file.

It may assist swirl's development if we could redesign course anatomy, from the top down, to take greater advantage of native R features, R package formatting, and existing CRAN infrastructure. For example, courses could be data-only packages, much like Hadley Wickham's recent nycflights13 and babynames packages (Wickham 2014c).

Course metadata could then be stored in a DESCRIPTION file, using a slightly modified set of key:value pairs to specify the course name, course description, version, keywords, package dependencies, and so on. The version field in particular could be very helpful for stricter management of content versioning, which is essentially ungoverned at the moment. Other R package conventions could be followed by placing lesson data in a data directory and any R code for initialization in the R directory.

Finally, by having some unique file at the top level of a course directory, we can imagine supplying a `find_available_courses()` function which searches GitHub for available swirl courses and presents a menu of options organized by subject matter to the user, right in the R console. The function could plug into GitHub's search API (GitHub 2014) to identify swirl courses based on this unique file. A feature like this might take some time to develop, but it is not inconceivable.

9.2 Web integration

9.2.1 Integrated web applications

The thing that makes swirl most unique among interactive platforms that teach people how to program is that it operates right in the R console. This means that users are exposed to an authentic programming environment and in order to use swirl, they must go through the process of installing the appropriate software (R, optionally RStudio, swirl, and other dependencies). However, we would

be silly not to take advantage of (or at least understand) some of the web-based tools that could enhance the learning process.

In particular, one technology that we have our eye on is Shiny (RStudio 2014c). Developed by RStudio, Shiny provides a framework for building web applications with R. Our vision is that we might use Shiny to build small, special-purpose web applications that run within or alongside swirl lessons. This would allow us to do many things that are currently impossible or prohibitively challenging to do in the R console. For example, we could display math notation using MathJax (MathJax Consortium 2014), interactive plots using ggvis, or interactive visual representations of code along the lines of Python Tutor (Guo 2013; Guo 2014).

We already encourage our users to install and use the RStudio IDE, since we believe it provides many useful features and makes for a much more user-friendly experience than the default R graphical user interface (GUI). There is a special window in the RStudio IDE called the Viewer Pane (RStudio 2014d), which acts as a miniature web browser and is capable of running local web applications.

We imagine users working through swirl lessons with the console open on one side of their screen and a combination of neatly integrated R scripts and web applications open on the other side of their screen. This sort of implementation would essentially turn the RStudio IDE into a full-fledged multimedia learning environment. Furthermore, because the web applications may run locally (provided local copies of any dependencies are available), this type of experience would not require the user to be connected to the internet.

9.3 Server-side evaluation

Another opportunity to integrate swirl with the web is that we could have students' responses to questions evaluated on a remote server, instead of locally on their machine. The main advantage to such an approach is that the logic used to determine whether a student's answer is correct or not would be hidden from the user. This is not currently the case, since swirl content is entirely open source, including the correct answers and answer testing mechanisms.

Because of the current arrangement, we emphasize the use of swirl as a practice tool more than an evaluation tool, the reason being that cheating is entirely possible if a student is so inclined. By exporting answer testing to a remote server, there would be no way for students to see either correct answers or the answer testing logic.

We have actually implemented a prototype of this workflow using Yhat's ScienceOps (Yhat 2014), which allows you to deploy R programs privately on their servers, then create an API to those programs. Essentially, we deploy answer testing code on Yhat, pass a user's response to it using the API, receive the response telling swirl whether the user's answer is correct or not, and have swirl react accordingly. There are other approaches to consider, but this seems to offer the most immediate solution.

Regardless of the approach we take, server-side evaluation of users' responses would require users to be connected to the internet while using swirl. We would never implement this type of feature across all content, since we think it's a great advantage that swirl can run entirely offline. Instead, it would be an additional feature of which those interested in using swirl for student evaluation could take advantage.

9.4 Instructor dashboard

We are aware of many instructors using swirl for their in-person courses. A common request is for an instructor dashboard, which would aggregate data on student completion and performance. The previous section on web integration hints at some ways we might approach this, but it is not yet clear what is the best alternative.

Ideally, such an instructor dashboard would facilitate classroom use of swirl by providing both student- and class-level data, useful visualizations and summary statistics, and perhaps a means of communicating with students. Instructors would log into the dashboard with a username and password and all data would be stored on a secure server.

It is quite possible that such an infrastructure would also allow for integration with outside learning management systems (LMSs) such as Blackboard or Moodle (Wikipedia 2014b). This could provide a means by which students would automatically receive credit in an external gradebook for completed swirl lessons, similar to what we have done with Coursera's gradebook API.

9.5 More "intelligent" answer tests

Something we are currently working on is improving the quality of swirl's built-in answer tests. These answer tests make up a library of functions available to swirl instructors for determining whether student responses meet the desired criteria and thus whether or not they are deemed correct.

Automatic evaluation is a challenging problem and developing these tools has been a tremendous learning experience for us.

One aspect of the R language that makes automatic evaluation particularly challenging is that there are often many different ways to achieve the same result. Any veteran R programmer can appreciate this. Perhaps the most classic example is that of the assignment operator. If someone wants to assign a value to an object in R, they can do so using any of the following: `<-`, `<<-`, `->`, `->>`, `=`, or `assign`. Our answer tests must be flexible enough to allow for stylistic differences, experimentation, and personal preference, but rigid enough to enforce overall correctness and good coding practices.

Recently, we have been experimenting with different ways of standardizing R expressions. For example, if we ask a user to sample five numbers from the values 1, 2, 3, ..., 10, without replacement, swirl should probably judge all of the following responses as being correct:

```
sample(1:10, 5)
sample(x = 1:10, size = 5)
sample(size = 5, 1:10)
sample(size = 5, x = 1:10, replace = FALSE)
sample(replace = FALSE, size = 5, 1:10)
```

All of the above expressions will accomplish the same result (although clearly some are more desirable than others), but R does not view the expressions equivalent. A more flexible answer testing approach should gracefully handle partial matching of argument names, unspecified arguments and their default values, numeric discrepancies (0.2 vs .2), and named arguments that are out of order. The `match.call` function in base R will play a prominent roll in addressing this problem.

9.6 Porting swirl to other programming languages

As swirl's popularity has grown, we've had several requests for implementations of swirl in other programming languages. Since swirl is written entirely in R and makes heavy use R's functionality for *computing on the language* (Wickham 2014d), it is not necessarily the case that it can be implemented in many other languages.

Python (Python Software Foundation 2014) is a very popular language in the scientific community. Some of swirl's developers have successfully rewritten key features of swirl entirely in Python 3, calling the result swirlypy. While it is still early days for swirlypy, the software, along with some

basic instructional content, is available for download and use from the primary author's GitHub repository (SashaCrofter 2014).

9.7 Translation to other spoken languages

We have heard from several swirl users interested in translating swirl into their native languages. There are two aspects of the swirl experience that would require translation: 1) the output and menu options that are currently hard-coded in English in the swirl R package and 2) any of the existing swirl courses. Of course, anyone can create new instructional content in their native language, but the effect would be limited if the menu options, etc. were not also available in the target language.

One of our developers has prototyped a feature that would allow swirl users to select their language of choice upon starting the program. The selection would cause swirl to display output and menu options in that language. We envision that, at least in the near future, course translations would be managed by simply creating a separate course for each language. Then, if a user desired, for example, the R Programming course in Chinese, they would just install the Chinese version of the course instead of the English one.

10 Conclusion

We have presented swirl, an open source software package that turns the R console into an interactive learning environment. swirl is used by students around the world who are looking for a fun and effective means of learning R programming and statistics. swirl's open architecture allows anyone to freely create and share interactive swirl courses. Lessons support a variety of question types, most important of which allow the user to interact directly with the R interpreter by entering commands at the prompt, while receiving helpful and immediate feedback from swirl.

While we are pleased with swirl's success, we have learned a tremendous amount since its inception and see many opportunities for improvement on the original design and the addition of new features. Future implementations of swirl will take advantage of native R infrastructure, focus on better performance, and incorporate more intelligent answer testing. We also plan to develop more tools for instructors who are using swirl in their classrooms, including a personalized and secure instructor dashboard. Interest in making swirl available in other spoken languages and programming languages will likely drive translation of both software and content over time.

11 References

Active State. 2014. "rappdirs GitHub Repository." http://cran.r-project.org/web/packages/rappdirs/.

Anderson, John R, and Brian J Reiser. 1985. "The LISP Tutor." *Byte* 10 (4): 159–175.

Barrow, Lisa, Lisa Markman, and Cecilia E. Rouse. 2008. "Technology's Edge: The Educational Benefits of Computer-Aided Instruction." Working Paper 14240. Working Paper Series. National Bureau of Economic Research. doi:10.3386/w14240. http://www.nber.org/papers/w14240.

Benjamin, Ludy T. 1988. "A History of Teaching Machines." *American Psychologist* 43 (9): 703–712. http://search.ebscohost.com/login.aspx?direct=true&db=pdh&AN=1989-06671-001&site=ehost-live&scope=site.

Code School. 2014. "Try R." http://tryr.codeschool.com/.

Code School LLC. 2014. "Code School Website." http://www.codeschool.com/.

Codecademy. 2014. "Codeacdemy Website." http://www.codecademy.com/.

Coursera. 2014a. "Coursera Website." https://www.coursera.org/.

———. 2014b. "Coursera Gradebook API." https://tech.coursera.org/app-platform/lti/.

DataCamp. 2014a. "Advanced R." https://www.datacamp.com/.

DataCamp. 2014b. "DataCamp - Becoming a Course Creator." https://teach.datacamp.com/faq.

Evans, Clark C. 2014. "The Official YAML Website." http://www.yaml.org/.

GitHub. 2014. "GitHub Search API." https://developer.github.com/v3/search/.

GitHub, Inc. 2014. "GitHub Website." https://github.com/.

Guo, Philip. 2014. "Python Tutor Website." http://www.pythontutor.com/.

Guo, Philip J. 2013. "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education." In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 579–584. SIGCSE '13. New York, NY, USA: ACM. doi:10.1145/2445196.2445368. http://doi.acm.org/10.1145/2445196.2445368.

Hornik, Kurt. 2014. "R FAQ." http://CRAN.R-project.org/doc/FAQ/R-FAQ.html.

JHU Data Science. 2014. "Johns Hopkins Data Science Specialization." https://www.coursera.org/specialization/jhudatascience/1.

Johnson, W. L., and Elliot Soloway. 1985. "PROUST: An Automatic Debugger for PASCAL Programs." *BYTE* 10 (4) (April): 179–190. http://dl.acm.org/citation.cfm?id=3351.3355.

Kulik, Chen-Lin C, and James A Kulik. 1991. "Effectiveness of Computer-Based Instruction: An Updated Analysis." *Computers in Human Behavior* 7 (1): 75–94.

MathJax Consortium. 2014. "MathJax Website." http://www.mathjax.org/.

Python Software Foundation. 2014. "Python Website." https://www.python.org/.

R Core Team. 2014. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org.

RStudio. 2014a. "The RStudio CRAN Mirror." http://blog.rstudio.org/2013/06/10/rstudio-cran-mirror/.

———. 2014b. "R Markdown." http://rmarkdown.rstudio.com/.

———. 2014c. "Shiny." http://shiny.rstudio.com/.

———. 2014d. "Extending RStudio with the Viewer Pane." https://support.rstudio.com/hc/en-us/articles/202133558-Extending-RStudio-with-the-Viewer-Pane.

SashaCrofter. 2014. "swirlypy GitHub Repository." https://github.com/SashaCrofter/swirlypy.

Skinner, B. F. 1954. "The Science of Learning and the Art of Teaching." *Harvard Educational Review* 24: 86–97.

———. 1958. "Teaching Machines." *Science* 128 (3330): 969–977.

———. 1986. "Programmed Instruction Revisited." *The Phi Delta Kappan* 68 (2): pp. 103–110. http://www.jstor.org/stable/20403280.

Sosa, Giovanni W., Dale E. Berger, Amanda T. Saw, and Justin C. Mary. 2011. "Effectiveness of Computer-Assisted Instruction in Statistics: A Meta-Analysis." *Review of Educational Research* 81 (1): 97–128. doi:10.3102/0034654310378174. http://rer.sagepub.com/content/81/1/97.abstract.

swirl Development Team. 2014a. "swirldev GitHub Organization." https://github.com/swirldev.

———. 2014b. "swirlify GitHub Repository." https://github.com/swirldev/swirlify.

———. 2014c. "swirl Course Repository." https://github.com/swirldev/swirl_courses.

———. 2014d. "swirl GitHub Repository." https://github.com/swirldev/swirl.

Tamim, Rana M., Robert M. Bernard, Eugene Borokhovski, Philip C. Abrami, and Richard F. Schmid. 2011. "What Forty Years of Research Says About the Impact of Technology on Learning: A Second-Order Meta-Analysis and Validation Study." *Review of Educational Research* 81 (1): 4–28. doi:10.3102/0034654310393361.

Treehouse Island Inc. 2014. "Treehouse Website." http://teamtreehouse.com/.

Wickham, Hadley. 2013. "frndly.R." https://gist.github.com/hadley/6734404.

———. 2014a. "Advanced R." http://adv-r.had.co.nz/.

———. 2014b. "appdirs GitHub Repository." https://github.com/ActiveState/appdirs.

———. 2014c. "New Data Packages." http://blog.rstudio.org/2014/07/23/new-data-packages/.

———. 2014d. "Non-Standard Evaluation." http://adv-r.had.co.nz/Computing-on-the-language.html.

Wikipedia. 2014a. "PLATO (Computer System) — Wikipedia, the Free Encyclopedia." http://en.wikipedia.org/w/index.php?title=PLATO_(computer_system)&oldid=624388372.

———. 2014b. "Learning Management System — Wikipedia, the Free Encyclopedia." http://en.wikipedia.org/w/index.php?title=Learning_management_system&oldid=615452003.

Yazdani, Masoud. 1986. "Intelligent Tutoring Systems: An Overview." *Expert Systems* 3 (3): 154–163. doi:10.1111/j.1468-0394.1986.tb00488.x. http://dx.doi.org/10.1111/j.1468-0394.1986.tb00488.x.

Yhat. 2014. "Yhat ScienceOps." https://www.yhathq.com/products/scienceops.

12 Author Biography

Nicholas (Nick) A. Carchedi was born and raised in Silver Spring, MD, USA. After earning a Bachelor of Science in Finance and Mathematics, he spent the next four years working in the financial services. Desiring to make a career change, Nick returned to school in 2013 to pursue a Master's degree (ScM) in Biostatistics at Johns Hopkins University. As a graduate student, his focus has been the intersection of statistics, data analysis, computing, technology, and education.